Apple Assembly Line

$1.50

Volume 2 -- Issue 11 August, 1982

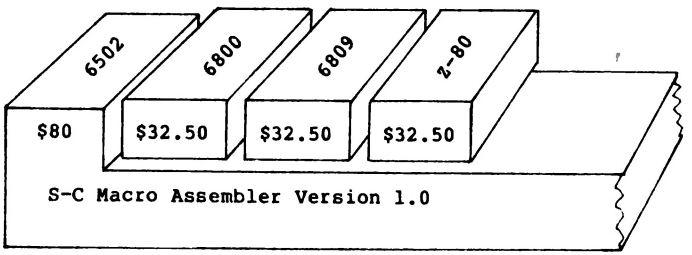
In This Issue...

Subscription Renewals

If your address label shows a number 8209 or smaller in the upper right corner, it is time to renew. That is $15 bulk mail in the USA; $18 First Class in USA, Canada, and Mexico; $28 to other countries.

New Macro Cross Assemblers Available

There are now three cross-assembler modules ready for the S-C Macro Assembler, and more to come. Each cross-assembler disk costs $32.50 to registered owners of the S-C Macro Assembler. You get both regular and language card versions, with documentation of the special features and differences.



Search and Perform Subroutine.....Bob Sander-Cederlof

When writing an editor or other single-keystroke command system, a very common need is a subroutine which branches according to the value of a character. In Pascal and some other languages there is even a special statement for this programming need: CASE. You might do it like this in Applesoft:

```
1000 GET A$
1010 IF A$ = "A" THEN 2000
1020 IF A$ = "C" THEN 3000
1030 et cetera
```

You will often find the equivalent code in assembly language programs:

```
1000         LDA CHARACTER
1010         CMP #'A
1020         BEQ CHAR.WAS.A
1030         CMP #'C
1040         BEQ CHAR.WAS.C
1050    et cetera
```

Of course, it frequently happens that the number of different values is small, and the code sequence above with several CMP-BEQ pairs is the most efficient. It loses a little of its appeal, though, when you have to do it for more than about ten different values. And what if the branch points are too far away for BEQ relative branches? Then you have to write:

```
1000         LDA CHARACTER
1010         CMP #'A
1020         BNE .1
1030         JMP CHAR.WAS.A
1040 .1      CMP #'C
1050         BNE .2
1060         JMP CHAR.WAS.C
1070 .2    et cetera
```

That takes seven bytes of program for each value of the character.

Personally, I like to put the possible values and the corresponding branch addresses in a table, and search that table whenever necessary. Each table entry takes only three bytes. If the subroutine is used with several tables, and if there are a lot of possible values, then the tabular method saves a lot of memory.

I used the tabular method in my still-in-development word-processor. To speed and simplify the coding of the table entries, I wrote a macro definition JTBL as follows:

```
1020         .MA JTBL
1030         .DA #$]1,]2-1
1040         .EM
```

S-C Macro Assembler.....$80.00
S-C Macro Cross Assembler Modules
    6800/6801/6802 Version.....$32.50
    6809 Version.....$32.50
    Z-80 Version.....$32.50
    Requires ownership of S-C Macro Assembler.
    Each disk includes regular and language card versions.

S-C ASSEMBLER II Version 4.0.....$55.00
Upgrade from Version 4.0 to MACRO.....$27.50
Source code of Version 4.0 on disk.....$95.00
    Fully commented, easy to understand and modify to your own tastes.

AAL Quarterly Disks.....each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982

Double Precision Floating Point for Applesoft.....$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).....(regular $79) $49.00
    Special price to AAL readers only, until 9/1/82!
Source Code for FLASH! Runtime Package.....$39.00

Super Disk Copy III (Sensible Software).....(reg. $30.00) $27.00

ES-CAPE: Extended S-C Applesoft Program Editor.....$40.00

Blank Diskettes (with hub rings).....package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.....$36.00

Ashby Shift-Key Mod.....$15.00
Paymar Lower-Case Adapter.....$37.50
    For Apples before Revision 7 only
Lower-Case Display Encoder ROM.....$25.00
    Works only Revision level 7 Apples. Replaces the encoder ROM.

Diskette Mailing Protectors.....10-99: 40 cents each
                                100 or more: 25 cents each
    Corrugated folder specially designed for mailing mini-floppy
    diskettes. Fits in standard 6x9-inch envelope. (Envelopes
    5-cents each, if you need them.)

Books, Books, Books.....compare our discount prices!
    "Beneath Apple DOS", Worth & Lechner.....($19.95) $18.00
    "Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36.00
    "Apple Graphics & Arcade Game Design", Stanton.....($19.95) $18.00
    "Assembly Lines: The Book", Roger Wagner.....($19.95) $18.00
    "What's Where in the Apple", William Leubert.....($14.95) $14.00
    "6502 Assembly Language Programming", Leventhal.....($16.99) $16.00
    "6502 Subroutines", Leventhal.....($12.99) $12.00
    "MICRO on the Apple--1", includes diskette.....($24.95) $23.00
    "MICRO on the Apple--2", includes diskette.....($24.95) $23.00
    "MICRO on the Apple--3", includes diskette.....($24.95) $23.00

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050   We take Master Charge and VISA ***

This defines a macro JTBL with two parameters. The first one will be the hexadecimal value to compare the test-character with, and the second one will be the branch address for that value. For example, if I write the macro call:

```
1400          >JTBL 86,FLIP.CHARS
```

the S-C Macro Assembler will generate:

```
                .DA #$86,FLIP.CHARS-1
```

The "-1" is appended to each branch address in the table, because I use the PHA-PHA-RTS method to perform the branch. Before I go any farther, here is the search and branch subroutine:

```
1220 SEARCH.AND.PERFORM.NEXT
1230          INY                   POINT TO NEXT ENTRY
1240          INY
1250          INY
1260 SEARCH.AND.PERFORM
1270          LDA T.BASE,Y          GET VALUE FROM TABLE
1280          BEQ .1                NOT IN THE TABLE
1290          CMP CURRENT.CHAR
1300          BNE SEARCH.AND.PERFORM.NEXT
1310 .1       LDA T.BASE+2,Y        LOW-BYTE OF BRANCH
1320          PHA
1330          LDA T.BASE+1,Y        HIGH-BYTE OF BRANCH
1340          PHA
1350          LDY #0      (SINCE MOST BRANCHES WANT Y=0)
1360          RTS                   DO THE BRANCH!
```

There are so far four different value-branch tables in my word processor. Here is an abbreviated listing:

```
1380 T.BASE
1390 T.ESC0 >JTBL 81,AUXILIARY.MENU
1400        >JTBL 82,SCAN.BEGIN
1410        >JTBL 83,TOGGLE.CASE.LOCK
 . . . . . .
1540        >JTBL 9B,ESC0.ESC
1550        >JTBL 00,SC.BELL
1560 *-------------------------------
1570 T.ESC2 >JTBL 81,AUXILIARY.MENU
 . . . . . .
1690        >JTBL EB,SCAN.RIGHT
1700        >JTBL ED,SCAN.DOWN
1710        >JTBL 00,ESC2.END
1720 *-------------------------------
1730 T.MAIN >JTBL C4,MAIN.DOS
1740        >JTBL C5,MAIN.EDIT
 . . . . . .
1800        >JTBL D3,MAIN.SAVE
1810        >JTBL 00,MON.BELL
1820 *-------------------------------
1830 T.AUX  >JTBL C3,COPY.BLOCK
1840        >JTBL C4,DELETE.BLOCK
 . . . . . .
```

```
1890          >JTBL D3,SAVE.SEGMENT
1900          >JTBL 00,SC.BELL
```

Notice that each of the four tables ends with a 00 value. The branch address after the 00 value tells where to branch if the current character does not match any values in the table.

When I want to compare the current character with entries in the T.MAIN table, here is how I do it:

```
2000          LDY #T.MAIN-T.BASE
2010          JSR SEARCH.AND.PERFORM
```

The LDY instruction sets Y to the offset of the table from T.BASE, and the search subroutine references the table relative to T.BASE. I use JSR to call the search subroutine. The search subroutine uses PHA-PHA-RTS to effectively JMP to the chosen branch address. And then the value processor ends with RTS to return to the next line after the JSR SEARCH.AND.PERFORM.

Counting all four tables, I have 45 branches, occupying 3*45 = 135 bytes. If I had used the CMP-BEQ method, which occupy four bytes per value, it would have taken 4*45 = 180 bytes. The subroutine is only 23 bytes long, so I saved 22 bytes. But if I needed the longer CMP-BNE-JMP sequences throughout, I would have had 7*45 = 315 bytes! Wow! Long live tables!

Tables have even more advantages. For one, they are a lot easier to modify when you want to add or delete a value. For another, the program is easier to read when there is no rat's nest of branches to try to unravel. For me, it almost makes the assembly listing as easy to read as the reference manual!

Notice that it would be possible to overlap tables using my subroutine. I might need at some times to search for 13 different values, and at others to search for only 7 of those same values, with the same branches. If so, the seven entries in common would be grouped at the end of the 13-entry table. The table has two labels, like this:

```
3000 T.13     >JTBL C1,DO.A
3010          >JTBL C4,DO.D
 . . . . .
3050          >JTBL CF,DO.O
3060 T.7      >JTBL C2,DO.B
3070          >JTBL C5,DO.E
 . . . . .
3120          >JTBL D7,DO.W
3130          >JTBL 00,DO.NOTHING
```

What about speed? Well, it is pretty fast too. The CMP-BNE-JMP takes five cycles for each value that does not compare equal, and finally seven cycles for the one which compares equal. If the tenth comparison bingos, that is 9*5+7 = 52 cycles. The subroutine takes 171 cycles for the same search. Over three times longer, but still less that 120 microseconds longer. You would have to perform the search over 8000 times in one day to add a whole second of computer time!

AUTO-MANUAL Toggle for S-C Macro Assembler.....R. F. O'Brien

Here is a small program to accompany Bill Morgan's Automatic Catalog in the June '82 issue of AAL. This routine adds an AUTO/MANUAL command toggle to the S-C Macro Assembler. Using CTRL-A when the cursor is at the beginning of a line enters the AUTO line numbering mode and waits for input of a line number and/or RETURN. Entering another CTRL-A while in AUTO mode and at the start of a line executes a MANUAL command.

In addition, I have added some code to provide slow and fast listings at a single keypress. CTRL-S does a SLOW LIST command, which is cancelled by a 'RETURN' during listing. CTRL-L will provide a listing at normal speed (assuming the slow list has been cancelled.)

The patch is implemented as follows:

1. Enter the S-C Macro Assembler
2. :$101D:33 N 1000G
3. :BLOAD AUTO/MANUAL PATCH
4. :$138D: 4C 28 32 (JMP PATCH instead of JSR BELL)
5. :BSAVE AUTO/MAN S-C MACRO ASM,A$1000,L$2300

Note: You may omit step 2 if you have already installed Bill's automatic CATALOG.

```
                    1000 *--------------------------------
                    1010 *  AUTO/MANUAL TOGGLE
                    1020 *
                    1030 *  BY ROBERT F. O'BRIEN
                    1040 *     14, CLONSHAUGH LAWN, DUBLIN 5.
                    1050 *--------------------------------
                    1060        .OR $3228
                    1070        .TF AUTO/MANUAL PATCH
                    1080 *--------------------------------
0024-               1090 CH           .EQ $24
11D2-               1100 SC.SLOW      .EQ $11D2
135E-               1110 SC.REENTER   .EQ $135E
13C3-               1120 SC.RETURN    .EQ $13C3
152A-               1130 SC.INSTALL   .EQ $152A
183F-               1140 SC.LIST      .EQ $183F
FF3A-               1150 MON.BELL     .EQ $FF3A
                    1160 *--------------------------------
                    1170 AUTO.MANUAL.COMMAND
3228- C9 81         1180         CMP #$81      CTRL-A?
322A- F0 0E         1190         BEQ AUTO.TOGGLE
322C- C9 8C         1200         CMP #$8C      CTRL-L?
322E- F0 41         1210         BEQ LIST
3230- C9 93         1220         CMP #$93      CTRL-S?
3232- F0 49         1230         BEQ SLOW.LIST
                    1240 *
3234- 20 3A FF      1250 BACK    JSR MON.BELL
3237- 4C 5E 13      1260         JMP SC.REENTER     BACK TO ASSEMBLER
                    1270 *--------------------------------
                    1280 AUTO.TOGGLE
323A- A5 24         1290         LDA CH
323C- C9 01         1300         CMP #1        BEGINNING OF LINE?
323E- F0 06         1310         BEQ AUTO.CMD
3240- C9 06         1320         CMP #6        AFTER LINE NUMBER?
3242- F0 16         1330         BEQ MANUAL.CMD
3244- D0 EE         1340         BNE BACK
                    1350 *--------------------------------
                    1360 AUTO.CMD
3246- A2 00         1370         LDX #0
3248- BD 55 32      1380 .1      LDA AUTO.TEXT,X   GET CHARACTER
324B- 20 2A 15      1390         JSR SC.INSTALL  PROCESS CHAR
324E- E0 05         1400         CPX #5
3250- 90 F6         1410         BCC .1
3252- 4C 5E 13      1420         JMP SC.REENTER
3255- C1 D5 D4
3258- CF A0         1430 AUTO.TEXT  .AS -/AUTO /
                    1440 *--------------------------------
                    1450 MANUAL.CMD
325A- A2 00         1460         LDX #0
325C- 86 24         1470         STX CH        GO TO START OF LINE
325E- BD 6B 32      1480 .1      LDA MANUAL.TEXT,X
3261- 20 2A 15      1490         JSR SC.INSTALL
3264- E0 06         1500         CPX #6
3266- 90 F6         1510         BCC .1
3268- 4C C3 13      1520         JMP SC.RETURN
326B- CD C1 CE
326E- D5 C1 CC      1530 MANUAL.TEXT .AS -/MANUAL/
                    1540 *--------------------------------
3271- A5 24         1550 LIST    LDA CH
3273- C9 01         1560         CMP #1        BEGINNING OF LINE?
3275- D0 BD         1570         BNE BACK      NO, RETURN TO ASSEMBLER
3277- 20 3F 18      1580         JSR SC.LIST
327A- 4C C3 13      1590         JMP SC.RETURN
                    1600 *--------------------------------
                    1610 SLOW.LIST
327D- A5 24         1620         LDA CH
327F- C9 01         1630         CMP #1
3281- D0 B1         1640         BNE BACK
3283- 20 D2 11      1650         JSR SC.SLOW   SET SLOW MODE
3286- 20 3F 18      1660         JSR SC.LIST
3289- 4C C3 13      1670         JMP SC.RETURN
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

Free Space Patch For the S-C Assembler.....Mike Sanders

Volume 5, Number 6 of Call A.P.P.L.E. has an article giving a DOS patch to replace the volume number printed during catalog with number of free sectors remaining on the disk.

The routine as published works for both Applesoft and Integer BASIC, but does not work with the language card version of the S-C Assembler. Only a few changes were needed to make it work with all three.

A call to Bob gave me the location of the decimal print routine in the S-C Macro Assembler, Language Card Version.

The original code as published in CAll A.P.P.L.E. checked location $E006 to see what language is in use. My code looks at $E001, which has a different value in each of the three:

Language	$E001
Applesoft:	$28
Integer BASIC:	$00
S-C Macro Assembler:	$94

The code in lines 1320-1370 checks which language is in use and jumps to the right routine. I also changed the zero page locations used to count the number of free sectors because the S-C Assembler print routine expects the two-byte value to be in $D3 and $D4.

The rest of the code works as explained in the Call A.P.P.L.E. article. I refer you to it for more details and as an excellent lesson on reducing the size of code.

Install the two patches to DOS by BLOADing the two binary files FREE.SECTORS.1 and FREE.SECTORS.2. The type CATALOG to see the how many free sectors you have.

```
                    1000 *SAVE S.FREE SECTORS
                    1010 *--------------------------------
                    1020 *      FREE SECTORS PATCH FOR DOS 3.3
                    1030 *--------------------------------
00D3-               1040 LOBYTE .EQ $D3
00D4-               1050 HIBYTE .EQ $D4
                    1060 *--------------------------------
B3F2-               1070 SECTOR.MAP .EQ $B3F2
E001-               1080 LANG.ID    .EQ $E001    LANGUAGE ID
E51B-               1090 PRT.INT    .EQ $E51B    INTEGER BASIC PRINT ROUTINE
ED24-               1100 PRT.FP     .EQ $ED24    APPLESOFT PRINT ROUTINE
DE00-               1110 PRT.SC     .EQ $DE00    S-C ASSEMBLER PRINT ROUTINE
                    1120 *--------------------------------
                    1130        .OR $BA69
                    1140        .TF FREE.SECTORS.1
                    1150 *--------------------------------
                    1160 FREE.SECTOR.PATCH
BA69- A0 C8         1170        LDY #$C8
BA6B- B9 F2 B3      1180 .1     LDA SECTOR.MAP,Y
BA6E- F0 0D         1190        BEQ .4              NO FREE SECTORS IN THIS BYTE
BA70- 0A            1200 .2     ASL                 SHIFT INTO CARRY
BA71- 90 FD         1210        BCC .2              SECTOR IN USE
```

```
BA73- 48        1220         PHA                 SECTOR FREE
BA74- E6 D3     1230         INC LOBYTE          COUNT IT
BA76- D0 02     1240         BNE .3
BA78- E6 D4     1250         INC HIBYTE
BA7A- 68        1260 .3      PLA                 SECTOR MAP BYTE AGAIN
BA7B- D0 F3     1270         BNE .2              IF ANY LEFT
BA7D- 88        1280 .4      DEY                 NEXT BYTE OF SECTOR MAP
BA7E- D0 EB     1290         BNE .1
BA80- A6 D3     1300         LDX LOBYTE          VALUE IN X AND A
BA82- A5 D4     1310         LDA HIBYTE
BA84- AC 01 E0  1320         LDY LANG.ID         CHECK WHICH LANGUAGE
BA87- 30 08     1330         BMI SCASM           $94: S-C ASSEMBLER
BA89- F0 03     1340         BEQ INTEGR          $00: INTEGER BASIC
BA8B- 4C 24 ED  1350         JMP PRT.FP          $28: APPLESOFT
BA8E- 4C 1B E5  1360 INTEGR  JMP PRT.INT
BA91- 4C 00 DE  1370 SCASM   JMP PRT.SC
                1380 *-------------------------------
                1390         .OR $ADB9
                1400         .TF FREE.SECTORS.2
                1410 *-------------------------------
ADB9- EA        1420         NOP                 FILLER
ADBA- A9 00     1430         LDA #0              ZERO THE COUNT
ADBC- 85 D3     1440         STA LOBYTE
ADBE- 85 D4     1450         STA HIBYTE
ADC0- 20 69 BA  1460         JSR FREE.SECTOR.PATCH
```

The Macro-Videx Connection.....Don Taylor

It seems that whenever I purchase a new hardware product for my Apple, I spend countless hours honing my most precious software tools to make them compatible with it. I purchased my Videx Videoterm card for use with Pascal, and had no intention of using it with the S-C Assembler. Then one fateful day I made a temporary patch to Version 4.0 -- just to see what it would look like -- and I was immediately hooked....

You won't believe what it's like to assemble with 80 columns of display! You can actually write source files that are legible on the screen, with no wraparound on comments -- even during assembly. What you see on the display is what you would see on a printer, only cleaner.

When I upgraded to the S-C Macro Assembler, I was compelled to produce a configuration file that would modify the new assembler to work with the Videoterm board. The resulting source file is included with this article.

The assembled SCM80 file will reconfigure a copy of the S-C Macro Assembler Version 1.0 that is currently resident in memory (for more about this concept, see "Controlling Software Configuration", AAL April '82).

Once the mods are installed you will be able to use your Videx for everything except: (1) Using the Escape-L sequence to LOAD a disk file whose name appears on the display, and (2) Using the copy key (right arrow). You will still be able to use Escape-L to generate the normal dashed comment line, and you can use the other escape functions to move the cursor and clear portions of the screen.

SCM80 will display control characters (and other selected strings intended to be so) in inverse on your screen, provided you have the standard (inverse) alternate character generator ROM installed in your Videoterm. If you have some other ROM installed, these characters and strings may be printed in Chinese. In this case you may want to modify the new character output routine!

SCM80 will also permit painless switching of case while using the assembler. A control-A keypress will always be recognized as a "shift lock" signal, while a control-Z will be treated as a "shift unlock". This feature makes it easy to write easy-to-read source files.

The assembled SCM80 code is moved into memory immediately following the assembler, and is located at one of two places, depending on which flavor (vanilla or language card) of the assembler you're using. The flavor of the configuration file is made to match that of the assembler through the use of a conditional flag (LCVERSION) and several conditional assembly statements. Another equate variable, SLOTNUM, allows you to specify the slot in which your Videx board resides.

There are two primary steps involved in installing the modified code in the assembler: (1) Moving the new code into the area of memory immediately following the assembler, and (2) Patching the existing assembler code to point to the new routines and then returning or cold-starting the system.

The SCM80 code contains both the new Videoterm support routines and the routines used to install those support routines. It loads in at $4000, stuffs the Videoterm routines just beyond the assembler code, and then performs the return or cold start. Depending on the flavor, a few other small tasks are performed in the process; let's take a closer look.

Lines 1280-1310 contain the two constants used to tailor SCM80 to assembler flavor and Videoterm slot number. The last two lines are the starting addresses where the new code will be relocated, depending on the flavor. The LCVERSION flag is used to determine the base address of the assembler in lines 1340-1380; this base address is used throughout the rest of the listing to determine absolute patch addresses within the assembler.

The Videoterm support routines are contained in lines 3240-3770. Lines 3400-3700 contain replacement routines for two of the routines in the line editor portion of the assembler. The NEW.WARM.ENTRY routine in lines 3240-3260 is intended to keep the Videoterm in the saddle during a RESET or system warm start.

The code in lines 3820-4740 are replacements for some of the standard monitor routines. Several of these routines have no other purpose than to support the escape cursor movements. In the case of the language card flavored RDKEY, an extra subroutine is provided to unprotect the RAM during case-shift sequences (more about that in a minute).

Lines 1770-2040 use the monitor's MOVE routine to slip the support routines into their designated origin at $3200 or $F400. The vanilla version patches the assembler's symbol table address to make room for the move; the language card version unprotects RAM prior to the move.

The patching of the assembler is done in lines 2050-2920. unused code is NOP-ed out here, and jumps are strategically poked in to point to the new routines. A replacement escape jump table created in lines 2950-3090 gets installed in the assembler, so the new escape routines can be accessed in the standard manner. The assembler's cold start routines are patched to point to the resilient NEW.WARM.ENTRY routine (more about that in a monent, too).

Lines 2870-2920 complete the installation and patching process. For the vanilla version, a simple RTS returns control to the calling program. The language card version first write protects RAM and then performs a DOS cold start. Once the assembled code has been installed and the patches made, the

installation portion of SCM80 is of no use, so a cold start should be performed to reset the assembler's file pointers, leaving only the SCM80 code that is now supporting your Videoterm.

Assembly and Installation

You'll note the absence of any .TF directive in the listing, meaning you'll have to manually save this file when you're done. This is because although the resulting object code will be located in continuous memory, it has origins (.OR directives) at two locations. The actual length of the file is calculated by a variable called LENGTH. The instructions for assembly are contained in the source file's title block. I call my vanilla patch file SCM80. and the language card version SCM80.LC.

With the assembler code resident in memory, there are several ways of installing the patches. Perhaps the most straightforward is to BRUN the assembled patch file, or BLOAD it and type 4000G as a monitor command. If you're using the vanilla assembler, you'll need to force a cold start of the assembler by typing "NEW" or 1000G as a monitor command; this action will ensure all the internal patches have been installed into DOS as well. The language card version cold starts itself, and requires no intervention.

A cleaner way is to use an EXEC file. The following file will bring up the vanilla version of the assembler:

```
REM LOAD ASM
CALL -151                 Enter the monitor
BLOAD S-C.ASM.MACRO       Load the Assembler
BLOAD SCM80               Load the patches
4000G                     Install them, and
1000G                     Start the assembler!
```

To load the language card patches with an EXEC file, refer to Bob's EXEC file on the top of Page 4 of the May '82 AAL, and replace "3D3G" with the following two lines:

```
BLOAD SCM80.LC            Load the LC patches
4000G                     Install them and cold start!
```

The character I/O is being vectored through routines at the end of the assembler; for the language card version, these routines are somewhere in $F4XX. If you decide to issue an "FP" command from that version, you'll find yourself in "Never-Never Land". It's good practice to issue a "PR#n" first (where "n" is the Videoterm's slot number). When you type "INT" to restart the assembler, the special I/O routines will automatically be hooked in.

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOL-KIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.
$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.
$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE..., CASE, SELECT,** and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.
$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output froms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.
$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.
$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.
$15 Disk, Applesoft (32K, ROM or Language Card).

(Add $4.00 for Foreign Mail)

***Apple II is a registered trademark of the Apple Computer Co.**

A Funny Thing Happened on the Way...

Bob thought it would be enlightening to touch on some some of the crazy things that went on during the development of these routines. I always marvel at people like Bob, Mike, Bill, and Lee, who have a gift for writing machine language, and can sit down and bang out a line editor in a few hours.

The rest of us aren't quite so fortunate. SCM80 took my three days to write, even though I had done some quick patches on Version 4.0. A couple of good ones popped up during that time, and I'll pass them along.

I was determined to interface the Videoterm using only its terminal functions, avoiding any internal Videoterm ROM routines that would make the interface version-dependent (my card matches neither the descriptions nor the ROM source listings contained in my manual!).

The Videoterm will not move its flashing cursor to a GOTOXY Location unless the cursor is first placed there and then a character is output; under BASIC, you can't just HTAB and VTAB to a position and GET a character -- you have to print a character first (even a null character will do it), in order to move the cursor!

After spending several hours fighting with the Videoterm over who was controlling the input and output cursor locations, I finally decided to designate my own locations for CH and CV (normally at $24 and $25) for use by the editing routines.

The other frustration I incurred was doing the case-switching in the replacement RDKEY routine. I was using the language card version, and had carefully checked my code, but the assembler just wouldn't switch case for me. True confession: it took almost fifteen minutes before it dawned on me that the assembler's case flag (at $D016) was write protected! Hence, the special unprotect subroutine called by the new RDKEY.

One final note concerns the contortions in the replacement COUT and WARM.ENTRY routines (at least I saw these coming!). We need to keep our new RDKEY routine in the DOS input hook to keep things working predictably. The Videoterm, when installed by placing it in the output hook and calling it to output a character, takes over the input hook as well. In addition, we have a replacement COUT routine that is designed to detect and modify control characters for display prior to their output.

In order to avoid arm-wrestling with the Videoterm over who controls the input hook, I used a strange but effective technique. During the installation and patch portion, I install the Videoterm in the designated slot, hook it in, and send a bogus character to make sure it has installed its warm entry I/O locations in DOS ($AA52-$AA56 for 48K machines). The code immediately following uses an internal assembler routine to calculate the address of the DOS output hook, regardless of memory size. The contents of the DOS output hook

are then moved into the new COUT routine, immediately following a JMP, and the same COUT routine is forced into the DOS hook, along with the new RDKEY routine. Whenever a character is output, it will first be given to COUT; when COUT has done its work, the character is then passed to the Videoterm's warm entry.

During the installation and patch, the warm start vector within the assembler was modified to point to the NEW.WARM.START routine, which re-installs COUT and RDKEY, keeping everything in sync. A RESET will always restore this condition, no matter what the Videoterm may have in mind!

The S-C Macro Assembler is a wonderful piece of software, and the upgrade is a steal at $27.50. The only thing that can top it is being able to use it with 80 columns of display!

If you find any errors in my patches, or come up with some new features, contact me at (206) 779-9508.

```
:
:LIST

 1000          .LIST OFF
 1010 *--------------------------------
 1020 *                  SCM80
 1030 *  Patches for S-C Macro Assembler V1.0
 1040 *         for Videx Videoterm Card
 1050 *
 1060 * Date: 7/10/82
 1070 *
 1080 * Don Taylor
 1090 * infoTool corporation
 1100 * Drawer 809, Poulsbo, WA  98370
 1110 *
 1120 * To assemble this file:
 1130 *
 1140 *       1.  Set SLOTNUM to slot number of videx card
 1150 *
 1160 *       2.  Set LCVERSION flag for
 1170 *             .EQ 1 for Language card version ($D000)
 1180 *             .EQ 0 for Standard version ($1000)
 1190 *
 1200 *       3.  Assemble as usual
 1210 *
 1220 *       4.  Use VAL LENGTH to get length in hex
 1230 *
 1240 *       5.  BSAVE SCM80, A$4000, L$LENGTH
 1250 *
 1260 *--------------------------------
 1270 *
 1280 SLOTNUM              .EQ 3     VIDEX slot
 1290 LCVERSION            .EQ 1     SCM80 version
 1300 PATCH.AREA           .EQ $3200
 1310 LC.PATCH.AREA        .EQ $F400
 1320 *
 1330 *--------------------------------
 1340          .DO LCVERSION
 1350 SCM.BASE    .EQ $D000
 1360          .ELSE
 1370 SCM.BASE    .EQ $1000
 1380          .FIN
 1390 *--------------------------------
 1400 *  Program Constants
 1410 *--------------------------------
 1420 MON.CSW              .EQ $36
 1430 MON.KSW              .EQ $38
 1440 MON.A1L              .EQ $3C
 1450 MON.A2L              .EQ $3E
 1460 MON.A4L              .EQ $42
 1470 SCM.POINTER          .EQ $58
 1480 SCM.CURR.CHAR        .EQ $61
 1490 SCM.ED.BEGLIN        .EQ $80
 1500 NEW.CH               .EQ $98
 1510 NEW.CV               .EQ $99
 1520 SCM.WBUF             .EQ $200
 1530 DOS.COLD.ENTRY       .EQ $3D3
 1540 DOS.IOHOOK           .EQ $3EA
 1550 FLAGS                .EQ $7F8  VIDEX Flag Byte
 1560 KEYBOARD             .EQ $C000
 1570 KEYSTROBE            .EQ $C010
 1580 SCM.WARM.ENTRY       .EQ SCM.BASE+$003
 1590 SCM.SHIFT.FLAG       .EQ SCM.BASE+$016
 1600 SCM.SYM.TABLE        .EQ SCM.BASE+$01D
 1610 SCM.TEST.DOS         .EQ SCM.BASE+$31E
 1620 SCM.RDL.EOL          .EQ SCM.BASE+$35E
 1630 SCM.RDL3             .EQ SCM.BASE+$3C3
 1640 SCM.ESC.TABLE        .EQ SCM.BASE+$467
 1650 SCM.ESC.L            .EQ SCM.BASE+$483
 1660 SCM.RDKEY.NO.CASE    .EQ SCM.BASE+$520
 1670 SCM.RDKEY.WITH.CASE  .EQ SCM.BASE+$4CA
 1680 SCM.SPC              .EQ SCM.BASE+$D92
 1690 MON.MOVE             .EQ $FE2C
 1700 MON.OUTPORT          .EQ $FE95
 1710 MON.COUT             .EQ $FDED
 1720 MON.RTS              .EQ $FF58
 1730 *--------------------------------
```

The most controversial computer magazine for serious Apple-Users.

To HARDCORE Computing

Isn't it time that you found out the true story behind the most controversial user-oriented computer magazine around today? Over 64 pages of facts, programming aids and program listings that includes columns on HOW TO: 1. Copy-protect disks, 2. Normalize "unlistable" programs, 3. Use bit-copy programs to make back-ups of the "uncopiables," 4. Write your own adventure-arcade games, 5. Market your software, and 6. Learn all about DOS.

"HARDCORE Computing ... warns pirates about the latest technology that companies are using against them." **TIME**, Feb. 8, 1982

"When some Apple enthusiasts ... heard about the boycott (of bit-copy ads), they concluded that it was nothing but censorship and another example of the magazines ignoring the average Apple user to placate their advertisers. So they started their own publication, HARDCORE Computing ..." **ESQUIRE**, Jan. 1982

HARDCORE COMPUTING
Dept. AL-1
P.O. Box 44549
Tacoma, WA 98444

Subscriptions:
$20.00 U.S.A. $28.50 Canada
$32.50 Mexico $42.00 Others

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

```
1740          .OR $4000
1750 START1   .EQ *
1760 *--------------------------------
1770 MOVE.CODE
1780          LDA #HERE
1790          STA MON.A1L
1800          LDA /HERE
1810          STA MON.A1L+1
1820          LDA #THERE
1830          STA MON.A2L
1840          LDA /THERE
1850          STA MON.A2L+1
1860 *--------------------------------
1870          .DO LCVERSION
1880          BIT $C083     Unprotect language card RAM
1890          BIT $C083
1900          LDA #LC.PATCH.AREA
1910          STA MON.A4L
1920          LDA /LC.PATCH.AREA
1930          STA MON.A4L+1
1940          .ELSE
1950          LDA #$33      Modify symbol table address
1960          STA SCM.SYM.TABLE
1970          LDA #PATCH.AREA
1980          STA MON.A4L
1990          LDA /PATCH.AREA
2000          STA MON.A4L+1
2010          .FIN
2020 *--------------------------------
2030          LDY #0
2040          JSR MON.MOVE
2050 INSTALL.PATCHES
2060          LDA #$EA      "NOP-OUT" unused code:
2070          STA SCM.BASE+$343
2080          STA SCM.BASE+$344
2090          STA SCM.BASE+$028
2100          STA SCM.BASE+$029
2110          STA SCM.BASE+$02A
2120          LDX #9
2130 .1       STA SCM.BASE+$298,X
2140          DEX
2150          BPL .1
2160          LDX #14
2170 .2       STA SCM.BASE+$4DE,X
2180          DEX
2190          BPL .2
2200          LDX #48
2210 .3       STA SCM.BASE+$B35,X
2220          DEX
2230          BPL .3
2240          LDA #$20                Install Videx during
2250          STA SCM.BASE+$295       cold start
2260          LDA #INSTALL.VECTORS
2270          STA SCM.BASE+$296
2280          LDA /INSTALL.VECTORS
2290          STA SCM.BASE+$297
2300          LDA #HOME     Patch clear screen routine
2310          STA SCM.BASE+$2A6
2320          LDA /HOME
2330          STA SCM.BASE+$2A7
2340          LDA #NEW.WARM.ENTRY     Set up warm start so
2350          STA SCM.BASE+$309       VIDEX card stays in.
2360          LDA /NEW.WARM.ENTRY
2370          STA SCM.BASE+$30A
2380          LDA #$10      Patch Escape Routine
2390          STA SCM.BASE+$486
2400          LDY #27
2410 .4       LDA NEW.ESC.TABLE,Y
2420          STA SCM.ESC.TABLE,Y
2430          DEY
2440          BPL .4
2450          LDA #$18      Modify MON.RDKEY jump addr
2460          STA SCM.BASE+$4D9
2470          LDA #$4C      Patch jump to new DISP LINE
2480          STA SCM.BASE+$B32
2490          LDA #NEW.E.DISP.LINE
2500          STA SCM.BASE+$B33
2510          LDA /NEW.E.DISP.LINE
2520          STA SCM.BASE+$B34
2530          LDA #80       Patch E.INPUT Routine
2540          STA SCM.BASE+$CA8
2550          STA SCM.BASE+$CAC
2560          LDA #NEW.CH
2570          STA SCM.BASE+$CB1
2580          LDA #NEW.CV
2590          STA SCM.BASE+$CB3
2600          LDA #VTAB
2610          STA SCM.BASE+$CB5
2620          LDA /VTAB
2630          STA SCM.BASE+$CB6
2640          LDA #SLOTNUM      Install VIDEX in hook
2650          JSR MON.OUTPORT
2660          JSR DOS.IOHOOK
2670          LDA #$8D          Send CR to get VIDEX warm
2680          JSR MON.COUT      entry point in DOS hook,
2690          LDY #8            then find warm entry
2700          JSR SCM.TEST.DOS              address
2710          LDY #1
2720          LDA (SCM.POINTER),Y
2730          STA FAKE.COUT+1   Save warm entry as normal
2740          INY               VIDEX COUT entry
2750          LDA (SCM.POINTER),Y
2760          STA FAKE.COUT+2
2770          LDA #COUT         Hook in new I/O routines
2780          STA MON.CSW
2790          LDA /COUT
2800          STA MON.CSW+1
2810          LDA #RDKEY
2820          STA MON.KSW
2830          LDA /RDKEY
2840          STA MON.KSW+1
2850          JSR DOS.IOHOOK
2860 *--------------------------------
2870          .DO LCVERSION
2880          BIT $C080     Write protect RAM
2890          JMP DOS.COLD.ENTRY
2900          .ELSE
2910          RTS
2920          .FIN
2930 *--------------------------------
2940 *
2950 NEW.ESC.TABLE
2960          .DA HOME-1
2970          .DA ADVNCE-1
```

```
2980          .DA BS-1
2990          .DA LF-1
3000          .DA UP-1
3010          .DA CLREOL-1
3020          .DA CLREOP-1
3030          .DA MON.RTS-1
3040          .DA MON.RTS-1
3050          .DA UP-1
3060          .DA BS-1
3070          .DA ADVNCE-1
3080          .DA SCM.ESC.L-1
3090          .DA LF-1
3100 *--------------------------------
3110 *  New routines to bind into the
3120 *  S-C Macro assembler
3130 *--------------------------------
3140 LENGTH1                .EQ *-START1
3150 HERE                   .EQ *
3160          .DO LCVERSION
3170          .OR $F400
3180          .ELSE
3190          .OR $3200
3200          .FIN
3210          .TA HERE
3220 START2                 .EQ *
3230 *--------------------------------
3240 NEW.WARM.ENTRY
3250          JSR INSTALL.VECTORS
3260          JMP SCM.WARM.ENTRY
3270 *
3280 INSTALL.VECTORS
3290          LDA #COUT
3300          STA MON.CSW
3310          LDA /COUT
3320          STA MON.CSW+1
3330          LDA #RDKEY
3340          STA MON.KSW
3350          LDA /RDKEY
3360          STA MON.KSW+1
3370          JSR DOS.IOHOOK
3380          RTS
3390 *
3400 NEW.E.DISP.LINE
3410          LDA SCM.ED.BEGLIN
3420          STA NEW.CV
3430          LDA #0
3440          STA NEW.CH
3450          JSR VTAB
3460          JSR SCM.SPC
3470          INC NEW.CH
3480          INC NEW.CH
3490          LDX #0
3500 .1       LDA SCM.WBUF,X
3510          BEQ .5
3520          ORA #$80
3530          CMP #$A0        Control char?
3540          BCS .2          No..
3550          AND #$7F        Flag it as inverse
3560 .2       LDY NEW.CH
3570          CPY #80         End of screen line?
3580          BCC .4          No..
3590          LDY #0          Set CH to beg of line
3600          STY NEW.CH
3610          LDY NEW.CV
3620          CPY #23
3630          BCS .3
3640          INC NEW.CV      No..
3650          BNE .4          ..Always
3660 .3       DEC SCM.ED.BEGLIN
3670 .4       JSR MON.COUT
3680          INC NEW.CH
3690          INX
3700          BNE .1          ..Always
3710 .5       JMP CLREOP
3720 *
3730 NEW.E.ZAP
3740          LDA #0          EOL mark
3750          STA SCM.WBUF,X
3760          JSR CLREOL
3770          RTS
3780 *--------------------------------
3790 *  Monitor Replacement Routines
3800 *--------------------------------
3810 *
3820 HOME     LDA #$8C        Send Form Feed Char
3830          JMP MON.COUT
3840 *
3850 CLREOL   LDA #$9D        Send CLEAREOL char
3860          JMP MON.COUT
3870 *
3880 CLREOP   LDA #$8B        Send CLEAREOS char
3890          JMP MON.COUT
3900 *
3910 ADVNCE   LDA #$9C        Non-destructive space
3920          JMP MON.COUT
3930 *
3940 BS       LDA #$88        Backspace
3950          JMP MON.COUT
3960 *
3970 LF       LDA #$8A        Linefeed
3980          JMP MON.COUT
3990 *
4000 UP       LDA #$9F        Reverse Linefeed
4010          JMP MON.COUT
4020 *--------------------------------
4030          .DO LCVERSION
4040 RDKEY    LDA KEYBOARD
4050          BPL RDKEY
4060          STA KEYSTROBE
4070          ORA #$80
4080          CMP #$81        Shift lock?
4090          BNE .1
4100          JSR UNPROTECT.LC.RAM
4110          LSR SCM.SHIFT.FLAG
4120          BPL .2          Return with errant key
4130 .1       CMP #$9A        Shift unlock?
4140          BNE .3          No, return with key
4150          JSR UNPROTECT.LC.RAM
4160          SEC
4170          ROR SCM.SHIFT.FLAG
4180 .2       BIT $C080       Reprotect LC RAM
4190          LDA #$96        Return with errant key
4200 .3       RTS
4210 *
```

```
4220 UNPROTECT.LC.RAM
4230         BIT $C083     Enable Bank 2
4240         BIT $C083
4250         RTS
4260         .ELSE
4270 RDKEY   LDA KEYBOARD
4280         BPL RDKEY
4290         STA KEYSTROBE
4300         ORA #$80
4310         CMP #$81      Shift look?
4320         BNE .1
4330         LSR SCM.SHIFT.FLAG
4340         BPL .2        Return with errant key
4350 .1      CMP #$9A      Shift unlock?
4360         BNE .3        No, return with key
4370         SEC
4380         ROR SCM.SHIFT.FLAG
4390 .2      LDA #$96      Return with errant key
4400 .3      RTS
4410         .FIN
4420 *-------------------------------
4430 *
4440 VTAB    LDA #$9E      Send GOTOXY char
4450         JSR MON.COUT
4460         CLC           Create ASCII x-posn
4470         LDA NEW.CH
4480         ADC #160
4490         JSR MON.COUT
4500         CLC           Create ASCII y-posn
4510         LDA NEW.CV
4520         ADC #160
4530         JMP MON.COUT
4540 *
4550 COUT
4560         PHA             Test for inverse
4570         PLA
4580         BMI FAKE.COUT   Not inverse: Take as is
4590         ORA #$80        Restore to "Normal" Apple ASCII
4600         CMP #$A0        Control char?
4610         BCS .1          No..
4620         ORA #$40        Yes: Make it printable
4630 .1      TAY             Save char
4640         LDA FLAGS+SLOTNUM
4650         PHA             Save flag byte
4660         ORA #1          Switch in alt char set
4670         STA FLAGS+SLOTNUM
4680         TYA             Get char back
4690         JSR FAKE.COUT
4700         PLA             Restore flag byte
4710         STA FLAGS+SLOTNUM
4720         RTS
4730 FAKE.COUT
4740         JMP $FFFF       Address will be fixed later..
4750 *-------------------------------
4760 LENGTH2              .EQ *-START2
4770 THERE                .EQ HERE+LENGTH2-1
4780 LENGTH               .EQ LENGTH1+LENGTH2
4790 *-------------------------------
4800         .EN:
```

Review of "Apple Graphics & Arcade Game Design"

If you are at all interested in Apple graphics, or writing animated hi-res games, this book is for you. Jeffrey Stanton, the author, may already be known to you. He is the editor of "The Book of Apple Software, and also has several Apple arcade games on the market. "Apple Graphics & Arcade Game Design" (AGAG) is 288 pages long, and retails for $19.95. (I am selling it for $18 plus shipping.) A coupon in the back enables you to purchase all of the source code shown in the book on diskette for only $15.

There are two parts to the book: first, a thorough explanation of Apple graphics, with numerous examples in both Applesoft and assembly language; second, design and programming of all the parts of a working arcade game.

AGAG is written for the advanced Applesoft or beginning assembly language programmer. You learn about both lo-res and hi-res graphics at the assembly language level. You learn the fundamentals, and then proceed to program scene scrolling, page flipping, laser fire, bomb drops, explosions, scoring, and paddle control routines. Sorry, nothing much about sound generation.

AGAG's pages are divided into 8 chapters as follows:

1. (25 pages) Applesoft Hi-Res
2. (34 pages) Lo-Res Graphics
3. (17 pages) Machine Language Access to Applesoft Hi-Res Routines
4. (23 pages) Hi-Res Screen Architecture
5. (36 pages) Bit-Mapped Graphics
6. (90 pages) Arcade Graphics
7. (44 pages) Games that Scroll
8. (5 pages) What Makes a Good Game

I noticed a few errors in the book: on page 149, flow chart lines are incorrectly drawn; on page 284, there is a large block of repeated text, and therefore possibly a missing block which should have been in the space. The word "initialize" is always incorrectly spelled "initilize". The index is very brief, only about 70 lines long; I believe it should be about 3 or 4 times longer to really help in locating items of interest.

Jeff does not seem to know about the existence of the S-C Macro Assembler. He repeatedly mentions the TED, Big-Mac, Merlin Assemblers, and occasionally refers to Lisa and DOS ToolKit. All the listings are in the Big-Mac format. You should have no trouble adapting them to the S-C format.

AGAG is an excellent tutorial. and includes many useful programs and ideas for anyone interested in Apple graphics. I heartily recommend the book, ranking it just under "Beneath Apple DOS" in importance and utility.

Quick Way to Write DOS on a Disk.....Bob Perkins

Tussy, OK 73088

I just received the July AAL and liked the little article on the "FILEDUMP" command. I had already done just about the same thing.

In fact, I make a lot of changes to DOS. Too many to POKE in every time I boot up. So I started looking around for a simple way to replace the DOS image on a disk without disturbing the programs already on it, and without using MASTER.CREATE. The July Call A.P.P.L.E. had a program to do it, only it seems much more complicated than my solution.

I used the S-C Macro Assembler to create a text file like this:

```
:1000  LOAD HELLO
:1010  POKE -21921,0:POKE -18448,0:POKE
-18447,157:POKE-18453,0:CALL-18614
:TEXT WRITE.DOS
```

Note the leading blank before the LOAD and the first POKE. It is there to leave room for Applesoft's "]" prompt.

Whenever I want to write the DOS image on a disk, I use the SHOW command to list out WRITE.DOS, and then trace over the two command lines from Applesoft. Presto-Changeo, a new copy of DOS goes out to the disk. I suppose you could even EXEC it, though I prefer to trace over it and haven't tried EXECing.

The LOAD HELLO is there to get the boot file name into DOS's filename buffer. You can use whatever filename you want, of course. POKE-21921 tell DOS that the last command was an INIT for its startup procedure (i.e. AA5F:00). POKE-18448 and -18447 start the write at 9D00 (B7F0:00 9D). POKR -18453 sets the expected volume number to zero, so a match to any volume will occur (B7EB:00). The CALL is to the "write DOS image" code inside DOS.

Correction

Last month I described the BIT instruction incorrectly. The next to the last paragraph on page 2 (in "Run-Anywhere Subroutine Calls") should read:

> The BIT instruction copies bit 7 of $FF58 into the N-status bit, and bit 6 into the Overflow status bit. This, in other words (since $FF58 has $60 in it) clears N and sets Overflow.

BIT does not affect Carry Status in any way. BIT also sets or clears the Z-status bit, according to the value of the logical product of the A-register and the addressed byte. If you want Z and/or N to be flags to the calling program, you will have to modify them after the BIT instruction.

Large Source Files and the S-C Macro Assembler.....Bill Morgan

One of the more common questions we get is: "How do I best use the .IN and .TF directives to handle very large programs?"

The main technique we use is the Assembly Control File (ACF), a short source file which is mostly made up of .IN statements to call the other modules. Here is an example, called SAMPLE.ACF:

```
1000        .IN SAMPLE.EQUATES
1010        .PG
1020        .IN SAMPLE.CODE.1
1030        .PG
1040        .IN SAMPLE.CODE.2
1050        .PG
1060        .IN SAMPLE.DATA
1070        .PG
```

SAMPLE.EQUATES is all the definitions for the program, SAMPLE.CODE.1 and SAMPLE.CODE.2 are the main body of the program, and SAMPLE.DATA contains all the variables and ASCII text. When you want to assemble the program, just LOAD SAMPLE.ACF and type MON C then ASM. The Macro Assembler will load each file and assemble it, in the order they are listed in the ACF. The "MON C" shows you the "LOAD file name" for each file, helping you to tell what's where.

Using this technique, a program can conveniently be broken into as many modules as you want, and can be as large as you want. The Macro Assembler itself is 26 source files on two disks! To spread the files across more than one disk, just add drive (and/or slot) specifiers to all the file names.

You can also use the ACF to do global search-and-replace operations on the entire program. Here are the commands to search SAMPLE for all occurences of the label MON.COUT:

```
:LOAD SAMPLE.ACF
:REP /        .IN/LOAD/A
:REP /        .PG/FIND "MON.COUT"/A
:TEXT COUT.SEARCH
:MON I
:EXEC COUT.SEARCH
```

This converts SAMPLE.ACF into an EXEC file that will list each occurence of "MON.COUT" in every module of the program. Here's what the file looks like now:

```
1000 LOAD SAMPLE.EQUATES
1010 FIND "MON.COUT"
1020 LOAD SAMPLE.CODE.1
1030 FIND "MON.COUT"
1040 LOAD SAMPLE.CODE.2
1050 FIND "MON.COUT"
1060 LOAD SAMPLE.DATA
1070 FIND "MON.COUT"
```

The ACF is also a good place for the .OR and .TF statements, comments about the assembly process, and any condition flags. Here is a more complicated version of SAMPLE.ACF:

```
1000 *--------------------------------
1010 *      SAMPLE FILE TO DEMONSTRATE ACF
1020 *--------------------------------
1030 LC.FLAG .EQ 0   =0 IF UPPER CASE ONLY
1040 *               =1 IF LOWER CASE VERSION
1050 *--------------------------------
1060         .OR $803
1070         .DO LC.FLAG
1080         .TF B.SAMPLE.LC
1090         .ELSE
1100         .TF B.SAMPLE.UC
1110         .FIN
1120 *--------------------------------
1130         .IN SAMPLE.EQUATES
1140         .PG
1150         .IN SAMPLE.CODE
1160         .PG
1170         .DO LC.FLAG
1180         .IN SAMPLE.LOWER.CASE.ROUTINES
1190         .PG
1200         .ELSE
1210         .IN SAMPLE.NORMAL.ROUTINES
1220         .PG
1230         .FIN
1240         .IN SAMPLE.DATA
1250         .PG
```

To use this ACF, just LOAD it, EDIT line 1030 to set LC.FLAG to 0 or 1, set MON C, and ASM. The Macro Assembler will load the appropriate source files for the version you want and direct the object code to the correct target file. To turn this ACF into an EXEC file for searching, you must delete lines 1000-1120, 1170, 1200, and 1230 before doing the REP commands.

For more information on the .IN and .TF directives, see pages 4-6 and 5-3/4 in the Macro Assembler manual. Conditional assembly is discussed on pages 5-9/10 and in chapter 7.

Another Customizing Patch for the S-C Macro Assembler

Version 4.0 of the S-C Assembler stopped after any assembly error. Many users requested that I modify it to continue to the end of assembly, and display the error count at the end. So I did.

Now some users are requesting that I change it back. They walk away during assembly, and the error messages scroll off the screen. (But you can put .LIST OFF at the beginning, and then only the error lines will list.)

There is a very simple patch for this. The byte at $1D6F ($DD6F in the language card version) is now $18. Change it $38 and assembly will stop after the first error message.

Patch for S-C Macro Assembler.....Bob Sander-Cederlof

When I added the lower-case options to the S-C Macro Assembler, I overlooked the fact that within .AS and .AT strings, and in ASCII literal constants, you would want lower case codes to be assembled. The assembler as it now is converts all lower case codes to upper case during assembly. For example, ".AS /Example/" would assemble all upper case ASCII, just as though you had written ".AS /EXAMPLE/"

The following patches will correct this problem, allowing you to specify lower case strings and constants when you wish.

```
$2961:EA EA EA EA EA EA

$31B8<1235.124BM

$1074:B8 31
$118C:B8 31
$11B2:B8 31
$187F:B8 31
$23FA:B8 31

$31CF:C8 84 7B C9 60 90 04 29 5F 85 61 60

$1240:20 CF 31

BSAVE ASM.WITH.LC.IN.AS,A$1000,L$21DB
(or whatever file name you wish)
```

The patches above are for the version which runs in mother-board RAM. The Language card version has different addresses, and you must first write-enable the language card. Assuming you are currently running the language card version, perform the patch as follows:

```
$C083 C083
$EAAD:EA EA EA EA EA EA

$F304<D235.D24BM

$D074:04 F3
$D18C:04 F3
$D1B2:04 F3
$D87F:04 F3
$E546:04 F3

$F31B:C8 84 7B C9 60 90 04 29 5F 85 61 60

$D240:20 1B F3

BSAVE LC.ASM.WITH.LC.IN.AS,A$D000,L$2327
(or whatever file name you wish)
```

Be aware that the above patches may conflict with other patches you may already have applied to your copy of the assembler. If you have already used the area from $31B8 through $31DB, or $F304 through $F326, you will need to use a different area and change the references accordingly.

Blinking Underline Cursor Routine.....Bill Linn

Early users of the ES-CAPE Applesoft Editing system (formerly known as AED II) have really come to appreciate the blinking underline cursor -- it simply doesn't tire the eyes as much as the standard flashing blank does. With the following subroutine, you can add this special touch to your own assembly language or BASIC programs!

The subroutine hooks into the monitor keyboard input vector at $38 and $39. Each time the monitor RDKEY subroutine is called, my KEYIN subroutine gets control. If the character on the screen at the cursor position is not an underline, I alternate the display of an underline and the original character every 1/4 second. If the original character was an underline, I alternate it with a blank. (If I alternate an underline with an underline, it is difficult to see anything happen!)

Lines 1210-1250 store the KEYIN subroutine's address in the keyboard input vector. When a request for a key press is made by an Applesoft INPUT command, for example, we get control at line 1270. The A-register has the current screen character. I save the A- and X-registers, because KEYIN must exit with the original values unchanged.

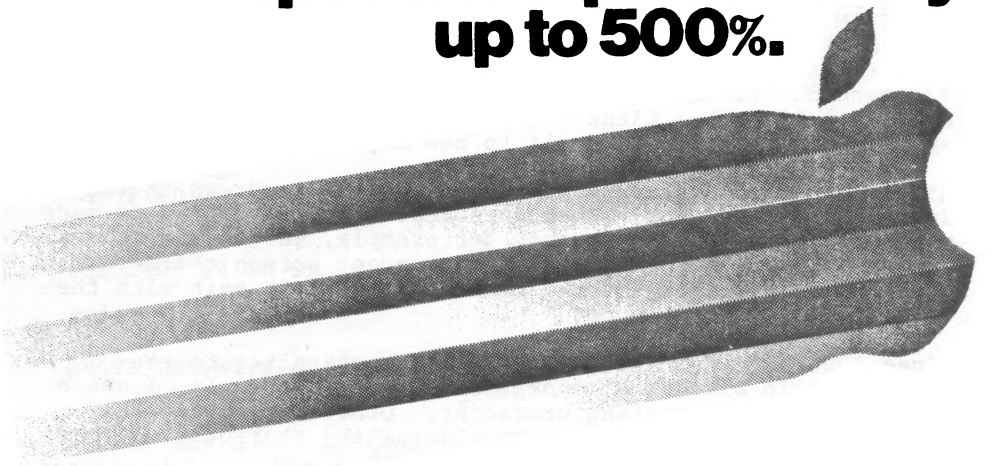
Lines 1290-1320 test the current screen character to see whether it is already an underline or not. If it is, I use a blank for the alternating character. Otherwise, I use the original screen contents for an alternating character. I push the alternating character onto the stack.

Lines 1330-1500 do the alternating. I look at the character on the screen: if it is an underline, I substitute the alternating character; if not, I store an underline. The lines 1430-1500 delay for about 1/4 second before the next alternation. If a keypress occurs, the loop ends by branching to ".5" at line 1540. You may wish to vary the blink rate by changing the value loaded into the Y-register at line 1430.

When a key is pressed we end up at line 1540, where I pop the alternating character off the stack. The I call the monitor bell subroutine for a short (10 half-cycles) bell. This makes an audible "click" for user feedback. (If you don't appreciate clicking keyboards, just delete lines 1550 and 1560.) Then I restore the Y-, X-, and A-registers to their orignal values, and jump into the monitor's KEYIN subroutine at $FD26. The monitor restores the original character to the screen, and returns with the keypress value in the accumulator.

I have set the subroutine origin to $300, but you can assemble it anywhere you like. In fact, it will run anywhere you put without reassembly, just so you load the correct address into $38 and $39 in the HOOK routine.

S&H Software presents Apple II™ users with two chances to increase speed and productivity up to 500%.



Universal Boot Initializer 4.0 will work up to 500% faster than standard Apple DOS 3.3...$69.95

The new UBI 4.0 now includes The DOS Enhancer, a DOS-transparent routine that allows execution of Apple DOS 3.3 files (Integer, Applesoft and Binary) up to 500% faster than standard Apple DOS—depending on file length. In addition, a new "FREE" command in DOS now allows determination of free space on a disk in "any slot, any drive"—from the command mode or the program mode.

UBI 4.0-created disks increase efficiency by breaking the "language barrier" between Apple II hardware and software and breaking the "time barrier" by loading the RAM card with FPBASIC/INTBASIC (or leading assemblers) in 1.7 seconds. This unique combination of features greatly increases productivity: copyable disks, one-stage booting with DOS 3.3 or DOS 3.2.1 PROMS, fast-loading the RAM card with the "missing BASIC" (or your favorite utility), fast-(B)RUNning or fast-(B)LOADing your programs and complete compatibility with all DOS 3.3 programs.

The new UBI 4.0 package includes the utility disk, training disk, support disk, demo disk and complete documentation. System requirements: Apple II or Apple II Plus, ROM or RAM card, DOS 3.3 or 3.2.1 and one or more disk drives.

Amper-Sort/Merge (A-S/M) works up to 500% faster than even VisiCorp's VisiFile™ program...$52.95

The fastest "file clerk" you've ever met. Of all the sort utilities developed to manage Apple II data files, none does the job nearly so fast as Amper-Sort/Merge.

Here's a quick profile of "A-S/M": With 25K of working memory, one of five unsorted files can be sort/merged into a single file of up to 125K per disk. If a file to be sorted is more than 25K in length, the utility temporarily lays it aside to be sorted and merged when more memory space is available.

Because sorting routines take up to 50% of the computer running time in many business applications, you'll reap continuing benefits having this "invisible speed demon" on your Apple II team. We estimate that it will save twenty to thirty minutes a day of your "human" clerk's time—time that would otherwise be spent waiting for "sort/merge" operations.

The A-S/M "speed demon" package includes the utility disk, the training disk and 24 page instruction manual. System requirements: 48K Apple II, ROM or RAM card, DOS 3.3 and one or more disk drives or 48K Apple II Plus, DOS 3.3 and one or more disk drives.

To Order: *Send Check To* **S&H Software, Box 5, Manvel, ND 58256 Credit Cards:** *Phone* **Cybertronics International Clearinghouse at 212 532-3089.**

S&H Software

Box 5 Manvel ND 58256
(701) 696-2574

After assembly, assuming it is origined at $300, you can BSAVE it with "BSAVE B.UNDERLINE,A$300,L$3C. Then to activate this routine from Applesoft, just BRUN the file B.UNDERLINE. All keyboard input through the standard monitor RDKEY subroutine ($FDOC) or Applesoft GET and INPUT statements will be prompted by the underline cursor. An "IN#0" will restore the familiar flashing blank. Have fun!

```
                        1000 *SAVE S.UNDERLINE CURSOR
                        1010 *-------------------------------
                        1020 *       BLINKING UNDERLINE CURSOR
                        1030 *       WRITTEN BY BILL LINN
                        1040 *-------------------------------
                        1050         .OR $300
                        1060 *-------------------------------
0024-                   1070 MON.CH      .EQ $24
0028-                   1080 MON.BASL    .EQ $28
0038-                   1090 MON.KSWL    .EQ $38
004E-                   1100 MON.RNDL    .EQ $4E
                        1110 *-------------------------------
FBE4-                   1120 MON.BELL2   .EQ $FBE4
FCA8-                   1130 MON.WAIT    .EQ $FCA8
FD26-                   1140 MON.KEYIN3  .EQ $FD26
                        1150 *-------------------------------
00A0-                   1160 BLANK       .EQ $A0
00DF-                   1170 UNDERLINE   .EQ $DF
                        1180 *-------------------------------
C000-                   1190 KEYBOARD    .EQ $C000
                        1200 *-------------------------------
0300- A9 09             1210 HOOK    LDA #KEYIN    SET INPUT HOOK
0302- 85 38             1220         STA MON.KSWL
0304- A9 03             1230         LDA /KEYIN
0306- 85 39             1240         STA MON.KSWL+1
0308- 60                1250         RTS
                        1260 *-------------------------------
0309- 48                1270 KEYIN   PHA              SAVE SCREEN CHAR
030A- 86 4E             1280         STX MON.RNDL      SAVE X-REG
030C- C9 DF             1290         CMP #UNDERLINE   IF CHAR ON SCREEN IS
030E- D0 02             1300         BNE .1             AN UNDERLINE
0310- A9 A0             1310         LDA #BLANK         THEN ALTERNATE WITH BLANK
0312- 48                1320 .1      PHA            SAVE CHAR TO ALTERNATE
                        1330 *-------------------------------
                        1340 *       ALTERNATE UNTIL KEY IS PRESSED
                        1350 *-------------------------------
0313- A9 DF             1360 .2      LDA #UNDERLINE
0315- A4 24             1370         LDY MON.CH
0317- D1 28             1380         CMP (MON.BASL),Y
0319- D0 02             1390         BNE .3
031B- 68                1400         PLA            GET ALTERNATE CHAR
031C- 48                1410         PHA            MAINTAIN ON STACK ALSO
031D- 91 28             1420 .3      STA (MON.BASL),Y
031F- A0 50             1430         LDY #80        80*256 BETWEEN BLINKS
0321- AD 00 C0          1440 .4      LDA KEYBOARD       KEY PRESSED?
0324- 30 08             1450         BMI .5         YES, CLICK AND RETURN
0326- CA                1460         DEX
0327- D0 F8             1470         BNE .4
0329- 88                1480         DEY
032A- D0 F5             1490         BNE .4
032C- F0 E5             1500         BEQ .2         ...ALWAYS
                        1510 *-------------------------------
                        1520 *       A KEY HAS BEEN PRESSED
                        1530 *-------------------------------
032E- 68                1540 .5      PLA            POP STACK ONCE
032F- A0 0A             1550         LDY #10        MAKE A "CLICK"
0331- 20 E4 FB          1560         JSR MON.BELL2
0334- A4 24             1570         LDY MON.CH
0336- A6 4E             1580         LDX MON.RNDL       RESTORE X-REG
0338- 68                1590         PLA            RESTORE ORIGINAL SCREEN CHAR
0339- 4C 26 FD          1600         JMP MON.KEYIN3
```

Review of QUICKTRACE.....Mike Sanders

I had already started writing my own debugger when I discovered QUICKTRACE; it was just what I needed and saved me all that work.

It has a good display that does not interfere with the normal Apple text screen. You can even trace code that sets the KSWL and CSWL switches and outputs to the screen. The tracing display takes the bottom four lines, but pressing the "P" key causes the normal bottom four lines to be displayed.

Tracing can be in one of three modes: single-step, trace, and background. Single-step and trace are what you would expect, analogous to the commands in the old Apple monitor ROM. Background turns off the display of executed instructions until a breakpoint occurs or the "ESC" key is pressed. This makes background the fastest mode.

Breakpoints can be set to stop when:

1. Any register or a memory location takes on a specified value.
2. An address or a range of addresses is referenced.
3. A specified opcode occurs.

QUICKTRACE can be BRUN at any point in memory and then called from your code by a JSR, or you can preset the QUICKTRACE program counter and start tracing at any location.

Subroutines can be executed at full 6502 speed (not traced). If you already know what the subroutine does there is no need to trace through it. Normally DOS calls are automatically done this way to prevent timing problems.

Overall I feel that QUICKTRACE is one of the five or so best programs I have ever purchased and no machine code programmer should be without it.

One feature not to be overlooked: QUICKTRACE is not copy protected.

QUICKTRACE was programmed by John Rogers and it is distributed by Anthro-Digital Software (formerly called Aurora Systems). It only costs $50.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; $18 per year sent First Class Mail in USA, Canada, and Mexico; $28 per year sent Air Mail to other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).